

# Resource Slayer

**Team members:** Colin Sullivan, Sarah Singhirunnusorn, Emmi Phillips, Vasu Agarwal

**Team Number:** 3

**Team Name:** PentaCode

**Project Name:** Resource Slayer

**Date:** 4/24/2024

### **1) Team Members:**

- Emmi Phillips: I'm a Computer Science major and my anticipated date of graduation is Fall 2025.
- Vasu Agarwal: My major is computer science and I will be graduating in the summer of 2024.
- Sarah Singhirunnusorn: I am majoring in Computer Science and minoring in Business, and I am set to graduate in the Spring of 2025.
- Colin Sullivan: I am majoring in Computer Science and minoring in Math, and I am set to graduate in Spring of 2026. I also co-developed a discord bot called RecNetBot in Python that currently has over 6k users. RecNetBot is a Discord bot that utilizes RecNet's API and is designed for quick data and stat lookups.

**2) Project Description:** For our project, we plan on building a single/versus resources management game where players will compete against one another to collect the most resources during a limited time frame. Within the game, players will start off by manually collecting resources, and they'll have the opportunity to spend their resources to build machinery to automate and speed up resource collection. Players will collect resources through farming various crops and exchanging those crops in the Store in order to grow their inventory. At the end, each player will receive bonuses for their performance during the game, and these bonuses have the potential to sway the outcome of the game. However, we are also considering rewarding the player when they reach a certain milestone within the game. For example, when they grow a certain amount of crops, they will be rewarded with some coins which could then be used to update their machinery.

### **3) Design:**

#### **a. Software Elements:**

- The software is composed of Classes, Inventory, Plant, PlantDataClasses, Store, dataStructure, entity, game\_2d\_resourceslayer, plantengine, plants, res/player, tile, and world. Classes include the GameMaster and one of the player classes. This is our main file package where the game is being mastered. Inventory includes the classes that have to deal with the player's inventory and the crops they have in their inventory. The first plant package includes which plants are plantable and how they are plantable. The PlantDataClasses control how the plants grow. Store was supposed to be where the player was supposed to sell crops and buy new crops. The dataStructure is where the coordinates for the player are. The entity package is where some of the collision game logic is for the player not to collide. The world package is where the tiles are marked where the player will collide with them. The game\_2d\_resourceslayer is where the gamePanel can be found.

The plant engine package controls when the plants pop up and where they can be planted. The plants package holds the corn xml file where we can find the instructions for the plant. The res package was a source file where the sprites were stored for all the different components. The tile package is where the tiles were implemented and loaded onto screen through a grid.

## b. Design

- i. Domain Model - overarching layout of the game's components
  - The Domain Model of our project defines and shows what components we need to create for the game as well as how each component created is interconnected. For instance, the GameMaster class defines numerous variables of the game such as the player (*player: player*), the different types of plants (*PlantFactory*), the world layout of the game (*gameWorld: ArrayList<ArrayList<WordTile>>*), and when the game is updated (*current tick: int*). These variables are then further defined and can be modified by the attributes and operations under their specific classes.  
**Specific parts:** Game, GameMaster, Player, ResourceManager, InventorySlot, PlantFactory. As well as simple defined classes, such as the PlayerItem, Plant, GameItem, and WordTile classes.
- ii. User Interface (GUI) - what the player uses to interact with the game
  - The player will use their mouse to click start to begin the game. The player will use different keys, for example, W, A, S, D, to make their character change positions. Our user interface will be defined in the player class. **Specific parts:** Player class, update position, Player coordinates, resources, plant, resource manager, level, look at UML for specifics
- iii. Text - informing players of the game's instructions within and pop-up text-box
  - The text will inform the player how to start the game. The text will display a pop-up that will display the amount of coins a player earns. **Specific parts:** LWJGL, drawTextBox()
- iv. Graphics - this is what appears on screen to the player and how the game looks
  - This project will utilize the LWJGL library as its main graphics library. LWJGL is a bare-bones abstraction of OpenGL, which allows much freedom in how we implement the graphical elements of this project. Basic actions such as drawing a tile or sprite will be defined as their own functions to make things easier. The graphics update loop will be detached from the logical game loop to allow for graphics to be updated as fast as the user's machine allows. **Specific parts:** LWJGL, look at UML for specifics
- v. Persistence - updating and storing the state of the game
  - Within this project, player data and game state needs to be stored, so that the player can return to the game as they left it after closing and returning

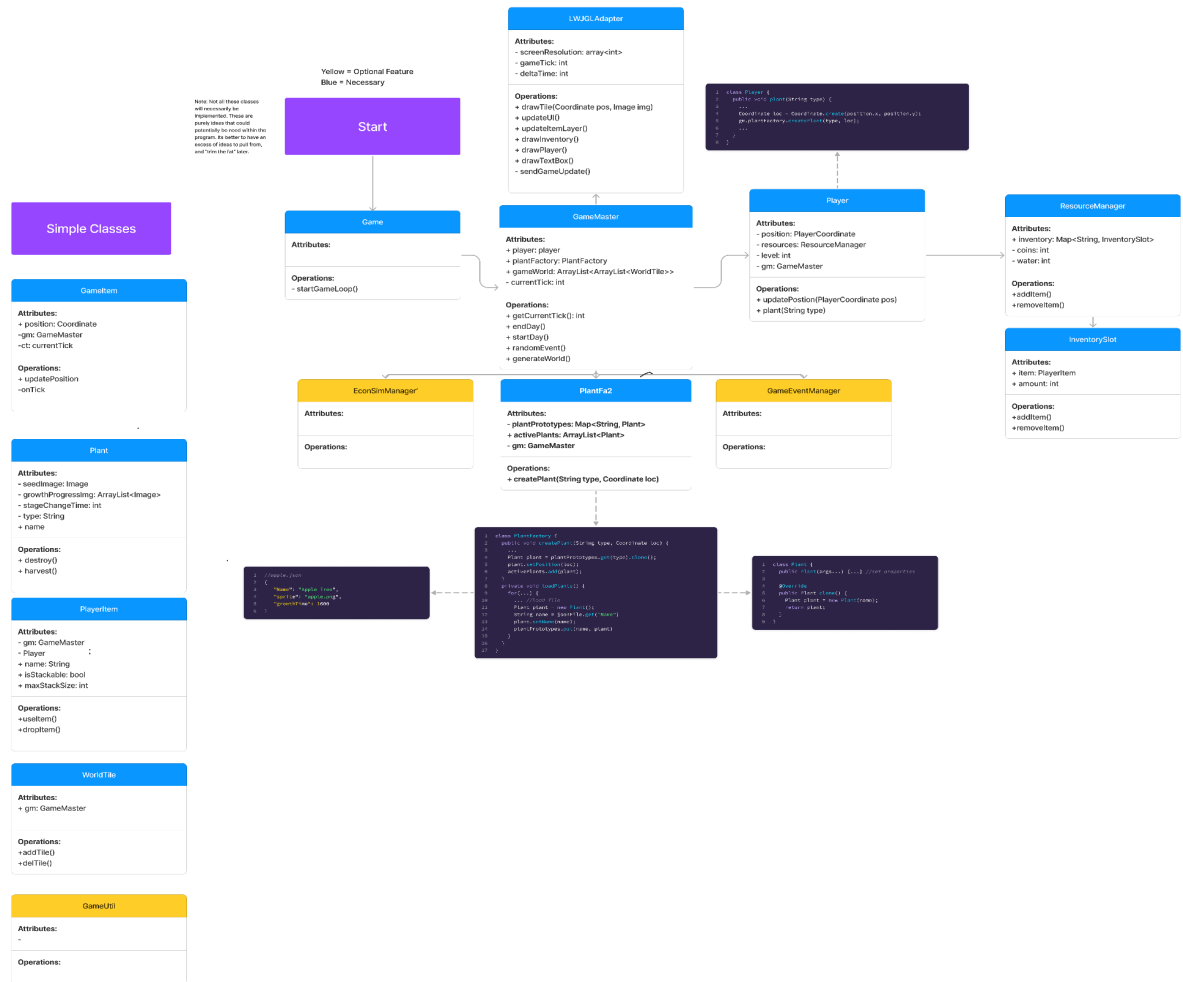
to the application. For the storage of this information, the application will utilize SQLite to store game data, as it is an efficient way of storing complex data that can be accessed and modified using familiar SQL commands. Storing player data will be quite simple, as most of the player stats are represented as integers. Saving the state of the inventory is a more complex matter, as items placed into the inventory will need to be loaded in the same layout as the player put them in. One potential solution to this problem would be to create an “*inventory*” table and include a “*position*” column that details specifically which slot the player put an item in, so that when data is being loaded the saved item can be placed in the same slot in the inventory. A similar principle can be used to store plant location, as another table could be used to store world information. **Specific Parts:** SQLite, SQLite Java Wrapper (any)

- vi. Editing and configuring the software product.
  - We went through a very intensive software editing process in which we edited the files for the components that we each created to make sure that they were synced to one another and ran properly. In addition, we also had some difficulties loading in the correct file path for the different sprites that we had to worktough when configuring the software product.

**c. Team Implemented: This is how our team implemented the Model-View-Controller architecture.**

- i. Model: Since the model represents the game’s data and logic, we created a UML file that showcased the classes for managing the game’s state, objects, and mechanics. For instance, the player class defines the player’s attributes.
- ii. View: We used Java’s graphical libraries (JavaFX or Swing) to render the graphics, draw the sprites, and handle the user keyboard inputs. For example, the KL class used the KeyListener interface to assign the WASD keyboard inputs to specific key events.
- iii. Controller: For the Controller aspect, we implemented classes that managed the player's input, processed the game events, and updated the game state. For example, the

d.



- [Link to UML :](https://www.figma.com/file/6SgNHqTC8zn7rL2u2tFQz5/Resource-Slayer?type=whiteboard&node-id=0%3A1&t=qUT08lpks0Ks6oS-O-1)  
<https://www.figma.com/file/6SgNHqTC8zn7rL2u2tFQz5/Resource-Slayer?type=whiteboard&node-id=0%3A1&t=qUT08lpks0Ks6oS-O-1>

#### 4) Implementation:

##### a. Design Implementation:

- The software is composed of Plant, PlantDataClasses, entity, game\_2d\_resourceslayer, plantengine, plants, res/player, tile, and world. Classes include the GameMaster and one of the player classes. This is our main file package where the game is being mastered. Inventory includes the classes that have to deal with the player's inventory and the crops they have in their inventory. The first plant package includes which plants are plantable and how they are

plantable. The PlantDataClasses control how the plants grow. Store was supposed to be where the player was supposed to sell crops and buy new crops. The entity package is where some of the collision game logic is for the player not to collide. The world package is where the tiles are marked where the player will collide with them. The game\_2d\_resourceslayer is where the gamePanel can be found. The plant engine package controls when the plants pop up and where they can be planted. The plants package holds the corn xml file where we can find the instructions for the plant. The res package was a source file where the sprites were stored for all the different components. The tile package is where the tiles were implemented and loaded onto screen through a grid.

**b. Packages and Classes:**

- i. Game\_2d\_resourceslayer:
  - 1. GAME\_2D\_RESOURCESLAYER.java - acts as a main class that sets up the JFrame
  - 2. GamePanel.java - regulates the frame rate at which the game updates and renders the graphics, defines the game loop for the game's execution, and draws the different game components to the screen.
  - 3. KL.java - class that implements the KeyListener interface and assigns the different keyboard inputs to specific key events.
- ii. entity:
  - 1. Entity.java - class that defines set variables used within the player and tile function.
  - 2. Player.java - class that defines the player's movement, loads in the player sprites, and controls how the player is drawn.
  - 3. CollisionBox.java - This detects where the player is going so that if they end up colliding with a block the player can still move in another direction.
- iii. tile: we had two classes:
  - 1. Tile.java: - class that defines the variables for a single tile
  - 2. TileManager.java - class that controls and loads in the sprites for the different tiles and assigns the type of tile to the specific index within the title map.
- iv. plant: within this package, we had 5 classes
  - 1. Plant.java - This class was responsible for holding all the data related to plants. It's built from the data in the plant xml file. It also renders the current state of the plant.
  - 2. PlantLoader.java - This class was responsible for loading all the plant data from the XML files.

3. PlantManager.java - This class implements a variant of the prototype creational pattern. Each plant class is created from data once, when a new plant needs to be created, it is copied from the prototype array.
  - v. PlantDataClasses:
 

Each of these files were auto generated using the JAXB binding in net beans. They contain a few fields to be inline with the plant XML schema.

    1. GrowthStages.java
    2. Harvest.java
    3. PlantData.java
    4. Seed.java
  - vi. Plants: package that contains all of the plant xml files.
  - vii. Utility:
    1. Time.java - class that defines the timing used for the animation of the different growth stages of the plant object.
  - viii. World
    1. World - Is connected to tile and tile manager and detects if the tile has a collision.
    2. WorldTile - Detects if the tile is plantable and sets positions for collidable tiles.
  - ix. GameMaster- This class controls the game progression, manages *player resources and triggers random events. It initializes the player, plant manager, and game world. It handles the tick management, simulates random events, and can generate the game world. It also provides a method to save the game state.*
- c. Third Party Libraries:
- i. Tile Sprites: [Download Cute Fantasy RPG - 16x16 top down pixel art asset pack by Kenmi - itch.io](#)
  - ii. Player Sprites: [Download Cute Fantasy RPG - 16x16 top down pixel art asset pack by Kenmi - itch.io](#)
  - iii. Plant Sprites: [Download Free Growing Plants Pack 32x32 by Danaida - itch.io](#)

## **5) Team Work:**

- a. **Team Process:** We first created a UML diagram that showcased all of the classes as well as their attributes and operations. This diagram gave us a foundation and framework to work with so that we had a better understanding of what direction to take when creating the game and how each component was connected to one another. Afterwards, we chose classes that we felt most confident coding. We would then upload these java files into GitHub. However, this became very inefficient when it came to actually connecting the

files. Thus, we assigned each team member a component to work on and then later implement those components together. For instance, Sarah was responsible for the game engine (ex: game loop, key listeners), while Colin worked on the game logic of planting the plants and collecting them. Once we had each component, Colin heavily worked on making sure that they were compatible with each other and would run properly.

- b. **Team Operations:** The team met 15 to 20 times in person and we met 3 times remotely. Most of the team had to get used to using gitHub because we had never used it before, but once we learned how to connect gitHub to NetBeans it proved to be very helpful. We shared all of our reports and documents over Google Drive. We communicated heavily over discord when we were not in person. Collin was over most of the game logic, Sarah was the leader over the game graphics, and Vasu led everything that had to do with the GameMaster class. Emmi and Colin made significant contributions to the collision controls within the World class and WorldTile class. Sarah made significant contributions to the Game Engine, Graphics classes, and implementing the sprites. Vasu made significant contributions to creating the xml files, GameMaster class, and formatting the basic classes. Colin made significant contributions in the collision controls, a majority of the game logic in almost every class, the implementation of planting the plants, and implementation delivery.
- c. **Time Budget:** Our original time budget consisted of 100 hours. We split these hours up into four quarters with 25 hours dedicated to creating a functioning game engine, 25 hours for the design and implementation of the game logic and its components, 25 for graphics related matters, and 25 hours for debugging and resolving any implementation issues. However, we had trouble efficiently distributing these hours over the semester and ended up rushing towards the end.
- d. **Problems:** During the beginning of the project, one of our group members, Zoe, ended up only coming to a couple of meetings. We tried reaching out to her on multiple platforms including Discord and directly messaging her. She never responded. Another problem we ran into was that our schedules did not align for us to spend a long amount of time working on this project together. We had to work on parts of this project separately until these last few weeks. We did make an effort to do the project during our meeting times every week. The team has had success in having the project come together in the amount of time we had. The graphics and game logic worked correctly in the end.

## **6) Exercise and Testing:**

- a. The team would run a live session over Visual Studio Code and NetBeans and then run all of the files that we have worked on. Individually we also ran each file we did on our



own to make sure they worked. The team would also play with the program to make sure it worked and didn't break each time we added something.

**b. Exemplary Use Case:**

1. User: Start the game
2. Player: Can move avatar with WASD.
3. Player: Walks to the tiles that look like dirt patches since these "farm" tiles have been designated as the only plantable tiles within the game
4. Player: Plants the seeds by pressing space while standing on the tile
5. Crops: After 15 seconds, the plant's growth stage changes instances and turns into a version that is harvestable
6. Player: Clicks space to harvest the crop
7. Game: Coin counter increases from the amount of crops collected from the plant.
8. Player: Walks to a different farm tile and plants a new seed there

**7) Best Work:**

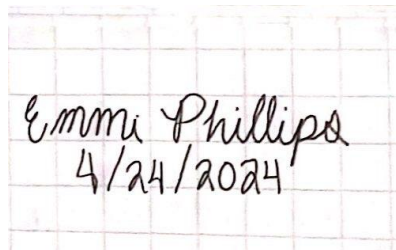
- a. The most interesting part of this project is the implementation of the dynamic import of plants and planting them on the specific tiles. Each plant and its related data is stored in a XML file. From this data, we built the java object that can create new instances of said plant within the game. This system allows us to add as many plants as we want, while the collision detection function allows us to only plant them on certain tiles.
- b. One of the main challenges we faced was syncing the time of the animation of the plant object in accordance to loading in the different instances of the plants after it's been planted on the tile.
- c. We think it's one of the team's best works because storing the plant data in different XML files was extremely efficient and the implementation method for the different instances was very clever and worked in an effective manner.

**8) Evaluation:**

- a. The initial description satisfied the project for the most part. We did make a farming simulator where when the crop is fully grown, the player earns money. As well, the player is manually collecting the crops.
- b. The quality of the project as a whole is very good for the time we had. The graphics, resolution, and the way the player moves is very impressive and works smoothly.
- c. The software could have improved by a lot of features that we wanted to extend onto the program. For example, the inventory of the player is in our program and we didn't have enough time to implement it. We could have added more components for the coins, inventory, and better graphics for the way we did certain things like planting crops.

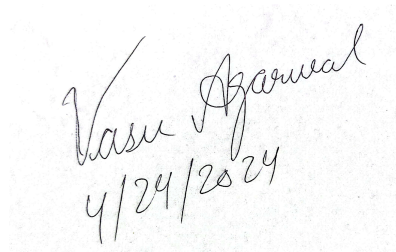
- d. The software could have had a store added to it where the player could sell their plants and buy the other crops we implemented into the program's code. The crops could have also been replaced with those new crops that we bought from the new store. The player would have had a bigger inventory to keep all the crops including apples, corn, wheat, and carrots. We also wanted to implement watering the plants and then uprooting them with a hoe. Also, we wanted to implement a time feature where the player would farm during the day and then at night the player's score would be calculated by how much they earned that day.

### 9) Submission Notice:



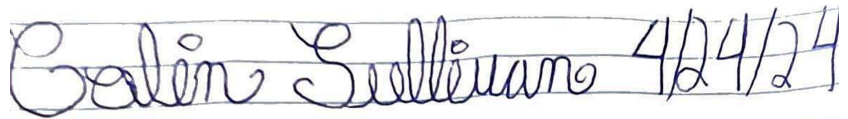
Emmi Phillips  
4/24/2024

- Emmi Phillips:



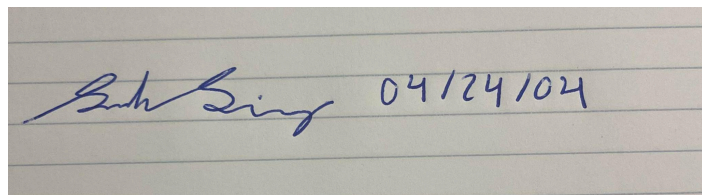
Vasu Agarwal  
4/24/2024

- Vasu Agarwal:



Colin Sullivan 4/24/24

- Colin Sullivan:



Sarah Sing 04/24/04

- Sarah singhirunnusorn:

